Overview of an Approach Describing Multi-Views/Multi-Abstraction Levels Software Architecture

Ahmad Kheir, Hala Naja, Mourad Chabane Oussalah, Kifah Tout

Overview of an Approach Describing Multi-Views/Multi-Abstraction Levels Software Architecture

Ahmad KHEIR[1,2], Hala NAJA[1], Mourad OUSSALAH[2] and Kifah TOUT[1]
[1]*LaMA Laboratory, Lebanese University, Tripoli, Lebanon*
[2]*LINA Laboratory, Nantes University, Nantes, France*
{*ahmad.elkheir, Mourad.oussalah*}*@Univ-nantes.fr,* {*hala.naja, kifah.tout*}*@ul.com.lb*

Keywords: Software Architecture: Viewpoints: Views: Abstraction Levels: Dependency: Consistency Rules.

Abstract: Views and abstraction levels are two major concepts introduced in the software engineering domain in order to enhance the architectural organization of complex systems' requirements. Despite numerous and substantial works that built and refined those concepts, they still less secure and less standardized to be part of a rough software architecture.
This paper begins with a survey of evolution of the role and usage of those concepts in software architecture, and ends with an overview of an approach integrating the views and abstraction levels concepts within a single rough multi-views/multi-abstraction levels software architecture and respecting their usage in an evolutionary architectural specification analysis.
The main benefits of this contribution was to allow system architects to solve more efficiently the complexity problems; and allow them to build a complex, organized and coherent architecture; and finally enhance the communication and harmony among different stakeholders.

1 INTRODUCTION

In large scale and complex systems, always different people are involved in the system development process and having for each of them different interests knowledge backgrounds and tools. Thus, these people, usually known as stakeholders, have their own look to the designed system, or their own viewpoints. Unfortunately, these stakeholders' viewpoints could be often complex and hard to be understood by the architect, which will be reflected as extra complexity in the system's architecture and difficulty to cover the entire system's domain. As mentioned in (Rozanski and Woods, 2011), *"a widely used approach is to attack the problem from different directions simultaneously. In this approach, the architectural description is partitioned into a number of separate views, each of which describes a separate aspect of the architecture.*" Informally, a view is an architectural design of a stakeholder's viewpoint that covers delicately all its interests and required functionalities, and represents them formally in a structured architecture. An abstraction level represents a level of details defined in order to treat the extra complexities retained after decomposing our system into multiple viewpoints, by relegating some viewpoints' details which appear to be irrelevant in the first stages of the modeling process to lower abstraction levels, so we can zoom into a specific viewpoint to the desired level of details.
In this paper, a survey is presented dealing with the evolution of role and usage of two major concepts in software engineering, which are the views and abstraction levels. Also an overview of our architectural modeling approach called *MoVAL* (Model, View and abstraction level) is suggested. In *MoVal*, both views and abstraction levels concepts are integrated allowing the architect to build robustly a multi-views and multi-granularities software architectures.
This paper is further structured as follows: In sections 2 and 3, studies and proposed solutions related to the views and abstraction levels fields are presented. Section 5 and 6 presents our comparative analysis and the deducted limitations of current solutions, respectively, in order to motivate the contribution of our approach presented in section 4, where we will suggest the beginning of theory and practice of views and abstraction levels suitable for software system architectures. Finally section 7 concludes the paper.

2 HISTORY OF VIEWS

Viewpoints are the perspectives of different people or stakeholders towards an information system (Finkelstein and Sommerville, 1996). Consequently, a view, which is a formal representation of an associated viewpoint, is an abstraction of the whole system's architecture suppressing some out-of-focus details.
Actually, this notion have been introduced in four different fields of the software engineering domain, which are: requirements engineering field, software systems modeling field, software architecture construction field and software systems implementation field.

2.1 Views in Requirements Engineering

One of the first and leading researchers that worked with viewpoint concept in requirements specification was A. Finkelstein in (Finkelstein and Fuks, 1989), where he and Fuks proposed a formal model aiming to construct system's specifications through a dialog in which the viewpoint negotiate. Robinson also worked on the requirements specification through dialog (Robinson, 1990), in 1990's and presented his tool, *Oz*, but he focused more on the representation of conflicting viewpoints and an automated way to produce resolutions.
Meanwhile, H. Delugash (Delugach, 1990) was working on the representation of each viewpoint's requirements in a conceptual graph in order to merge them by analyzing those conceptual graphs and performing a join so that we obtain a single coherent set of requirements for the target system.
In 1994, B. Nuseibeh et al. have introduced in (Nuseibeh et al., 1994), the relationships that could exist between viewpoints (overlapping, complementary, contradiction) in requirements specification and have proposed a framework to describe those relationships.
Finally, in 1997, Sommerville have contributed in this domain in (Sommerville and Sawyer, 1997) by proposing, *Preview*, a model for requirements discovery, analysis and negotiation.

2.2 Views in Software Systems Modelling

Prior to the incomparable commonness of UML in software systems modeling field, Mullery has developed a method, called CORE, for requirements specification and design in 1979 (Mullery, 1979). CORE is a modeling approach supported by diagrammatic notation which decompose the modeling process to many steps, and define for each of them a definition level and a set of viewpoints for which they will model their requirements and establish, in an iterative way, the consistency among them.
In the early 1990s, Booch et al. have developed the *Unified Modeling Language* (UML), which was adopted by the OMG in 1997. In the current version of UML, 13 types of diagrams, where each diagram type has an implicit viewpoint. Several researchers have worked to extend UML model in order to add the concept of dynamic views and viewpoints like in (da Rocha, ; Nassar, 2003).
In 2003, a view-based UML extension, VUML (Nassar, 2003), have been introduced to offer a UML modeling tool in which the multiple viewpoints of the system are taken into account in the modeling phase, so they offered a methodology to construct multiple class models one for each viewpoint and merge them to obtain a single VUML model describing all the viewpoints. Then in 2009 (Nassar et al., 2009), they proposed a code generation tool that generates object code of the corresponding VUML model. Finally, they worked on the automation of the composition process of VUML model (Anwar et al., 2010; Anwar et al., 2011) and included the attributed graphs in order to reach this goal.
Also one of the important works done in this field was (Dijkman et al., 2008) where they presented a framework allowing the architect to model a system from different perspectives or viewpoints by defining a collection of basic concepts common to all viewpoints and defining for each viewpoint its associated level of abstraction for its basic concepts. In this work they focused on the definition of consistency relationships between the views.

2.3 Views in Software Architecture

In Software Architecture, various models have been proposed of how to create a documentation (i.e. an architectural description) by the separation of concerns. Each model describes a set of viewpoints and identifies the set of concerns that each of them address. The different models cover the same software architecture domain, including the associated structural, organizational, business and technological environment. The concept of Views appears in one of the earliest papers, Perry and Wolf's classic (Perry and Wolf, 1992) on Software Architecture in the early 1990s. In (Sowa and Zachman, 1992), the approach presents an extensive set of constructs called *columns* which is very similar to views. In 1995, Philippe Krichten proposed four different Views (Kruchten, 1995) of a system and the use of a set of scenarios (use cases) to

check their correctness. In (Hilliard, 1999), the author proposes a Architecture Description Framework (ADF) in which views are first-class entities governed by type-like entities called viewpoints characterized in terms of a set of properties pertaining to their application, a viewpoint language. This study was the basis of the IEEE Standard 1471 (Group, 2000) which has formalized concepts used in Software Architecture and brought some standardization of terminology used in this field. It recognized the importance of Views in architectural description and adopted the Viewpoint concept defined earlier in (Hilliard, 1999). Definitions of Terms *Views* and *Viewpoints* as given in the Standard are at follows :

- **View :** is a representation of a whole system from the perspective of a related set of concerns.
- **Viewpoint:** is a specification of the conventions for constructing and using a view. A pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis. A viewpoint is a realization of the concerns of one or more stakeholders.

Here, it is useful to review some of the approaches, which are based on the above definitions of Views and Viewpoints, for their salient characteristics:

- In (Rozanski and Woods, 2011), the authors propose a Viewpoint Catalogue for Information Systems, extending the 4+1 set of viewpoints identified by Kruchten (Kruchten, 1995) comprising 6 core viewpoints including : the Functional, Information, Concurrency Development, Deployment and Operational.
- In (Clements et al., 2002), 3 viewpoints, called viewtypes, are identified which are : Module, Component-and-Connector and Allocation. Also, the author defines a three-step procedure for choosing the relevant views for a system based on stakeholder concerns. Also, this approach uses the concept of combined views and prioritization to bring the view set into manageable size for real-world projects.
- the valuable survey done in (May, 2005) compares between several view-based models and tries to find out the correspondencies between the different views proposed and the divergence between them. It proposes an optimum framework coverage encompassing a viewpoint set selected from different models with the greatest coverage of the framework concepts. The limitation of this approach is that despite the selected views are not independent, no connections or relationships are defined between views.

2.4 Views in Software Development

In Software Development field, the Aspect-oriented development techniques consider that a number of software development *concerns* could not be handled using the modularization boundaries inherent in object-oriented languages and propose new artifacts (beyond method, class or package) to separate new kinds of concerns that tend to be amalgamated in object-oriented paradigms (Mili et al., 2006). In this area, several methods are proposed :

- the Subject-Oriented Programming (SOP) (Ossher and al., 1995) technique addresses the *functional requirements*. It views object oriented applications as the composition of several application slices representing separate functional domains. Each slice called *subject* consists of a self-contained, object-oriented program, with its own class hierarchy. Then, a composition language is defined allowing the composition of class hierarchies (subjects).
- the Aspect-Oriented programming (AOP) technique, such in (Kiczales et al., 1997) and (Majumdar and Swapan, 2010), addresses the *non-functional requirements*, including architectural aspects, error handling, security, distribution, persistence, etc. It defines *aspect* as an implementation of a concern that pertain to several objects in a collaboration. An aspect is a code module addressing specific concern and that cross-cut various components in an application.
- the View-Oriented programming technique (Mili and al, 1999) considers each object of an application as a set of core functionalities available, directly or indirectly, to all users of the object, and a set of interfaces specific to particular uses, and which can be added or removed during run-time.

Hence, we found that also in software development, researchers always try to apply this concept of views decomposition, presented in many forms (subjects, aspects and views) in order to benefit from its organization and complexity resolution advantages.

3 HISTORY OF ABSTRACTION LEVELS

Abstraction levels is a core concept in software engineering, it is a way to deal with software systems' complexities giving the architect or the analyst the ability to examine different topics of the system at different levels of details (i.e. abstraction levels) according to the purpose.

The abstraction levels was defined by Jeff Kramer in (Kramer,) as *"a cognitive means according to which, in order to overcome complexity at a specific stage of a problem solving situation, we concentrate on the essential features of our subject of thought, and ignore irrelevant details."*
Actually there were not so many researches in this domain as it was the case in viewpoints domain, but among the valuable works done in this field (Regnell et al., 1996) could be considered, where the authors have proposed a hierarchical modeling approach that consists of three static distinct abstraction levels:

- The environment level, which could be seen similar to the scenario view of UML.
- The structural level, in which they decomposed each use case of the precedent level to multiple units named *episodes*.
- The event level, in which episodes are described in further details through a *Message Sequence Chat* annotation.

In (Medvidovic et al., 1996), authors have proposed a software architecture's modeling process of component based architectures, composed of four modeling process abstraction levels or, actually, four modeling layers, which are: architectural components specification, components' interfaces definition, architectural description and architectural styles rules. Actually, they used in their work the pipe-and-filter and the C2 architectural styles as illustrations.
In (Monperrus et al.,), authors have proposed a decomposition approach of meta-model to multiple abstraction levels. In this approach they defined a lower abstraction level as a specialization of the upper abstraction level and proposed splitting rules that are based on the inheritance and specialization relationships of the meta-model.

4 MoVAL APPROACH

MoVAL (Model, View, Abstraction level) is an architectural modeling approach that comply to the IEEE recommended practice for architectural descriptions (Group, 2000) with some additions in order to assist the system's architect solving some types of complexities that was not been treated in the existing approaches, and give him additional flexibility in the development process comparing to those approaches. So, MoVAL was designed in order to contribute in complex systems architectural modeling and development process independently of the adopted development technology. Thus, the viewpoints concept was involved in order to reduce the system's complexities

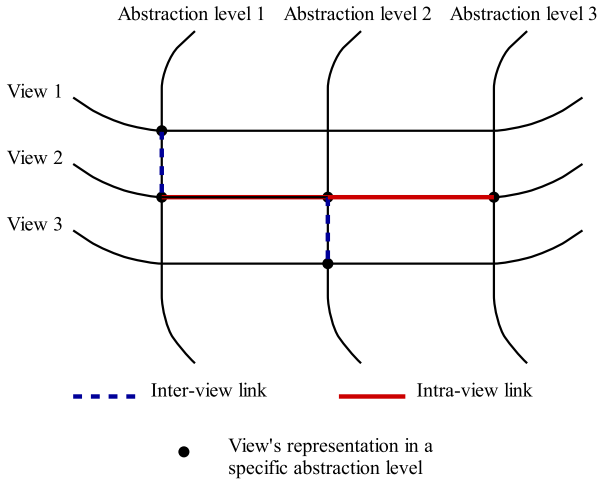

Figure 1: Matrix representation of the MoVAL model

by splitting vertically its requirements. In addition, the abstraction levels concept was involved in order to allow the system architect to also split horizontally the system's requirements into many abstraction levels, and to allow other participants in the development process to move between those abstraction levels according to the purpose of the current task.
Hence, the main benefits of MoVAL is to allow system architects to solve more efficiently the complexity problems; and allow them to build a complex, organized and coherent architecture coordinated in a set of models related one to the other via links holding the coherence constraints; and finally enhance the communication and harmony among different stakeholders by giving each of them the appropriate tools to express his interests.
Actually, a MoVAL model could be represented, as shown in figure 1, by a matrix defining its views and abstraction levels, and illustrating for each view either it's represented or not in each abstraction level. Also, this matrix illustrates, where exists, the link between two view's representations, either belonging to two different views and the same abstraction level (inter-views link), or belonging to the same view and different abstraction levels (intra-view link). Those links actually will hold the consistency rules that will be defined later in our approach, and that will ensure the consistency of the resulting software architectural description.

Each column of the matrix represents a distinct view of the model, and each line represents an abstraction level defined for this model. In reality, a view of the model does not have necessarily representations in all of the defined abstraction levels due to the absence of the isomorphism between model's views. So, each view could have one or more rep-

resentations respectively in one or more abstraction levels defined by the architect, depending on the system's requirements.
Note that the system decomposition to views and abstraction levels should be done, for now, manually based on the architects' expertise in the system's application domain, and there is no defined mechanisms offered in MoVAL helping architects to determine the architectural views and abstraction levels.

4.1 Views in MoVAL

A view is a representation of the entire target system considering a specific sub-set of the system's concerns that are directly related to the view's associated viewpoint. This viewpoint could be considered as a specification of the conventions for constructing and using the view or a pattern or template from which to develop this view. Consequently, a view in a MoVAL architecture represents the vision of its associated stakeholders and its associated requirements or concerns. Those concerns, as defined by Sommerville in (Sommerville, 2007), as the *statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations*.
Actually, even though each viewpoint (consequently a view) is often related to a specific category of stakeholders, but always it's up to the architect to select the appropriate viewpoints and views to be considered in the architecture, and it's up to him also to decide if those viewpoints will be indirectly related to different stakeholders.
Thus, a MoVAL architectural description always contains a set of views that are connected one to the other from certain points via well-defined connectors, the links.
For example, for a stock management system, we can identify the merchants' viewpoint and some of its concerns:

- A merchant shall be able to monitor his stock at any time.
- A merchant shall be able to refill his stock by making orders from the stock of his associated sub-distributor.
- A merchant shall be able to make sale operations on items of his own stock.

We can identify also the web developer's viewpoint and some of its concerns that could overlap some of the merchants' concerns:

- A merchant shall be able to make sale operations on items of his own stock.
- Merchant's sale operation shall be done in a secured way using the https protocol.

4.2 Abstraction Levels in MoVAL

An abstraction level represents a level of details defined by the system architect and included in the MoVAL architecture in order to reduce the complexity of the problem and to relegate some details, which appear to be irrelevant in the first stages of the modeling process, to lower abstraction levels. For now, this definition should be done in MoVAL manually by describing for each abstraction level its level of details or what to be included in the views at this abstraction level, and what to be relegated to other abstraction levels.
For example, for the stock management system introduced above, we could identify three levels of details for this model, which are:

- The global data, presenting only the definition of the main entities of the related view.
- The stock data, presenting the stock elements in the vision of the concerned actors.
- The transaction data, in which we define also the supported transactions of each view point implementing this abstraction level.

From another side, an abstraction level is meant to accept some specific formalisms, specifying the associated modeling languages and tools (UML, MDA, AGM, etc ...), in which some participants in the development process might define its associated representations. Note that a formalism could be dependent to a specific technology, called in this case concrete formalism, like the class model formalism of UML which is dependent to the object oriented programming technology. Also, a formalism could be independent of any technology like the use case model formalism of UML, in which we can represent a set of requirements formally and independently of the adopted technology, in this case the formalism is called abstract formalism.
Normally, an abstraction level can be eventually consistent with a subset of the architecture's defined views but not necessarily all the views, what gives birth to a set of architectural representations associated each to a specific abstraction level and aggregating together the views of MoVAL architecture.

4.3 Architectural Representations in MoVAL

An architectural representation is *a view and abstraction level dependent*. It is a group of models repre-

senting a specific architectural view in a specific abstraction level. This architectural representation could be considered by itself as a sub-system that could be described and modeled, separately of all the architectural description process, by a set of models. Architectural representations are illustrated in figure 1 by a black point inside the model's matrix.
In general, the views of a MoVAL architecture do not have an isomorphism among them, so each of them could have architectural representations associated to a subset of the architecture's defined abstraction levels, not necessarily all of those abstraction levels, by the mean that each architectural view could have a different number of architectural representations, and each abstraction level would not be necessarily associated to all of the defined architectural views.

4.4 Links in MoVAL

In MoVAL approach, relations between architectural representations are defined. They connect the architectural views among them. Those relations are represented via the *Links*. A link is a simple architectural connector having some properties, like the source and destination architectural representations, a semantic role and a semantic link. It holds the dependency, dominance and consistency rules that might be organized and described in the architectural description.
In MoVAL, two distinct categories of links are distinguished:

- The *Intra-View links*, having the source and destination architectural representations belonging to the same architectural view.
- The *Inter-Views links*, having the source and destination architectural representations belonging to two different architectural views.

4.5 MoVAL Meta-model

Actually, MoVAL meta-model extends the IEEE standard 1471-2000. Thus we have kept some definitions like the definition of a system, architecture, architectural description, stakeholder, viewpoint, view, concern and model, and we have equally added some other definitions like the definition of an abstraction level, architectural representation, formalism, and link.
A *System*, as it was defined in the IEEE standard 1471-2000, is not limited to individual applications but it encompasses them to cover also the subsystems, systems of systems and all kind of software interests' aggregations. A system always has different categories of *Stakeholders*, which are the participants in every phase of his life cycle. They could be individuals, teams or even organizations interested in this system, like the system architects, developers, analysts, experts contributing in the system development, users, etc. ...
Each of those stakeholders focuses on a specific part of the system requirements saturating his interests. Hence, those interests of different stakeholders are defined as different sets of *Concerns* overlapping in certain cases and contradicting in other cases.
In order to be constructed, a system might be decomposed to a set of *Models* representing each, formally, a part of that system. This set of models and the way they are related is called the system's *Architecture*, described and documented in an *Architectural Description* (AD) in order to allow different stakeholders to understand the structure of the system.
Each AD comprises different *Views* (Section 4.1), and each of these views conforms to a *Viewpoint* specifying the set of concerns that might be considered in the view's definition, and specifying also the candidate *Formalisms* that could be used in the model definition.
Normally, the viewpoint is not the only factor restricting the candidate formalisms that could be used in the model definition but also the abstraction level (Section 4.2) for which this model belongs should accept the selected formalism.
In addition, to illustrate the multi-hierarchy, or the multi-abstraction levels aspect of an architectural description in MoVAL, the hierarchy or the decomposition of an architectural view to different *Architectural Representations* was defined, and those architectural representations were associated to different abstraction levels and related one to the other via the *Intaview Links*, if the architectural representations belong to the same view, and via the *Inter-views Links*, if the architectural representations belong to two different views (Section 4.4).
Figure 2 represents the proposed meta-model.

5 ANALYSIS

Table 1 presents an elicitation of five main characteristics of several approaches sharing similar purposes with MoVAL's approach.

- The inter-views relations support column gives idea about the considered approach, if it represents the relations that exist between the views, and what types of relations it represents.
- The stakeholder roles column defines the considered set of stakeholders for each approach.

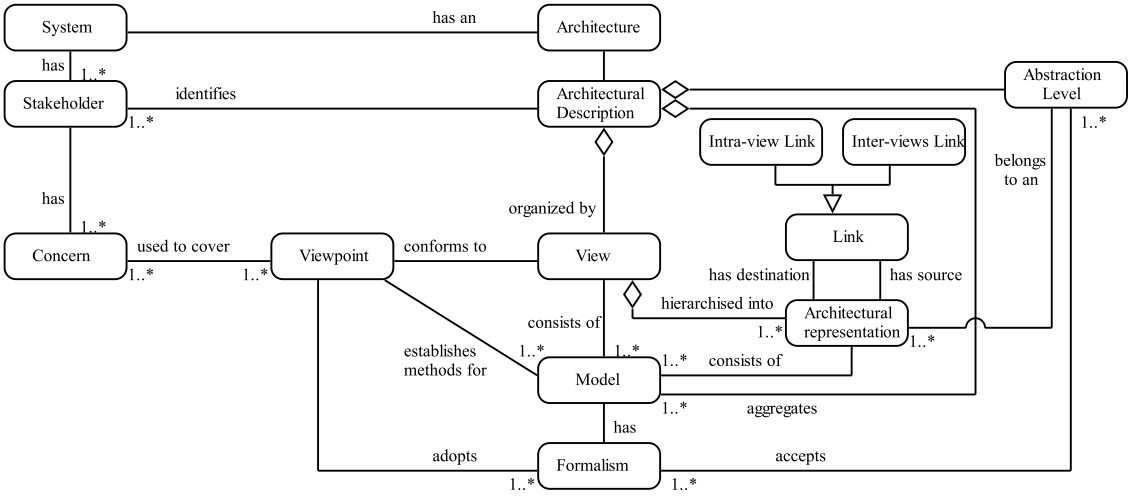

Figure 2: Conceptual model of MoVAL

- The Fixed/Not Fixed views column indicates either if the number of considered views in each approach is fixed or not.
- The hierarchy support column tells if each approach represents multiple levels of details/ abstraction levels/ hierarchy levels and gives a hint about its support if exists.
- Finally the methodology support column indicates for each approach either it offers, or not, a methodology or process to be applied on a real case in order to obtain the resulting model or architecture.

Actually, this table illustrates the fact that some approaches do not represent the relations that may exist between different viewpoints, marked as Not Available (NA) in the table, and some other approaches admit the existence of some kind of relations (overlapping relations in (Sommerville and Sawyer, 1997; Mullery, 1979) and consistency relations in (Hilliard, 1999; Group, 2000; Rozanski and Woods, 2011)) but also do not represent them formally. Other approaches represent special kinds of relations between views as in (Delugach, 1990), where they represent formally the dependency and overlapping relations between different views.
In addition, the hierarchy support column indicates, for each approach, if it has support for hierarchy levels, but unfortunately, as mentioned in table 1, all of those approaches do not give a definition for such levels, except for (Dijkman et al., 2008) in which authors have associated, for each viewpoint, a single and specific abstraction level, and in (Mullery, 1979) in which author have defined the steps in his approach which hold in reality some kind of levels of details.

6 RELATED WORKS LIMITATIONS

As mentioned in (Rozanski and Woods, 2011), using views and viewpoints to describe the architecture of a system benefits the architecture definition process in a number of ways that are : *Separation of concerns*, *Communication with stakeholder groups*, *Management of complexity* and *Improved developer focus*. However, some limitations remain when using existing view-based approaches. We can summarize some of these limitations, which were solved in MoVAL, as follows:

- *Needs to move between different abstraction levels :* Despite software architecture is defined as the high-level system structure, we assume that a software architect should define views at different levels of details. In fact, some stakeholders may settle for an overview information, while others require detailed information and those whose needs are between (i.e. require some details). So, the software architect needs to think in terms of *different abstraction levels* and to be capable to move between abstraction levels. Actually, in all the studied related works, an architect cannot specify a view or one of its models in multiple abstraction levels, however in MoVAL he could identify for each view as many abstraction levels as he needs, like the example mentioned in section 4.2.
- *Lack of an Architectural Description Process:* In almost all the studied approaches, except in (Clements et al., 2002), it is unclear what the software architect has to do to define the suitable architectural description. This task is based mainly

Table 1: Existing works and their main characteristics.

	Inter-Views Relations Support	Stakeholder Roles	Fixed/Not Fixed Views	Hierarchy Support	Methodology Support
(Finkelstein and Fuks, 1989; Robinson, 1990)	NA	End user	Not fixed	NA	Yes
(Delugach, 1990; Nuseibeh et al., 1994)	Dependency, Overlapping	Development participants, End user	Not fixed	NA	Yes
(Sommerville and Sawyer, 1997)	Admits overlapping	Development participants, End user	Not fixed	NA	Yes
(Mullery, 1979)	Admits overlapping	Analyst, End user	Not fixed	Steps	Yes
(Nassar, 2003)	NA	End user	not fixed	NA	Yes
(Dijkman et al., 2008)	Refinement, Overlapping	Development participants, End user	Not fixed	Single abstraction level per viewpoint	Yes
(Sowa and Zachman, 1992)	NA	Planner, Owner, Designer, Builder and Subcontractor	Fixed (Data, Function, Network, People, Time and Motivation)	NA	No
(Kruchten, 1995)	NA	Development participants	Fixed (Scenario, Logical, Development, Process and Physical)	NA	No
(Hilliard, 1999; Group, 2000; Rozanski and Woods, 2011)	Admits consistency relationships existence	Development participants, End user	Not fixed	NA	No
MoVAL	**Formal consistency relationships**	**Development participants, End user**	**Not fixed**	**Multiple abstraction levels per viewpoint**	**Yes**

on his (her) experience and skills. So, MoVAL approach aims at defining an Architectural Description Process (ADP) which guides the software architect while defining the software architectural description. The ADP should be flexible, non-constraining and iterative. Actually, MoVAL's ADP is out of this paper's focus and won't be presented.

- *Views Inconsistency:* Using a number of Views to describe a system inevitably brings inconsistencies problems. So we need to achieve a cross-view consistency within an architectural description, which is not offered by the majority of the studied related works, but MoVAL has defined the links (section 4.4) which hold the needed coherence rules in order to solve those inconsistencies problems.

7 CONCLUSION

This paper has presented a preliminary approach for documenting intensive software systems architecture based on views and abstraction levels. This approach complies with the view-based approach of IEEE-1471 and aims at extending it in three different ways that are : (1) providing the software architect means to define views at different levels of detail and to move between them; (2) defining relationships that solve the inconsistencies between the architectural views; (3)

the specification of an architectural description process which guides the architect while selecting pertinent views, documenting them and refining them at different abstraction levels.
Actually, we are validating MoVAL's methodology or development process that will allow system architects to build a rough and coherent multi-views/multi-abstraction levels software architecture.

REFERENCES

Anwar, A., Dkaki, T., Ebersold, S., Coulette, B., and Nassar, M. (2011). A formal approach to model composition applied to VUML. In *Engineering of Complex Computer Systems (ICECCS), 2011 16th IEEE International Conference on*, page 188197.

Anwar, A., Ebersold, S., Coulette, B., Nassar, M., and Kriouile, A. (2010). A rule-driven approach for composing viewpoint-oriented models. *Journal of Object Technology*, 9(2):89114.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., and Stafford, J. (2002). A practical method for documenting software architectures.

da Rocha, E. S. Guidelines for business modeling elaboration based on views from domain information.

Delugach, H. S. (1990). Using conceptual graphs to analyze multiple views of software requirements.

Dijkman, R. M., Quartel, D. A. C., and van Sinderen, M. J. (2008). Consistency in multi-viewpoint design of enterprise information systems. *Information and Software Technology*, 50(7):737752.

Finkelstein, A. and Fuks, H. (1989). Multiparty specification. In *ACM SIGSOFT Software Engineering Notes*, volume 14, page 185195.

Finkelstein, A. and Sommerville, I. (1996). The viewpoints FAQ. *BCS/IEE Software Engineering Journal*, 11(1):24.

Group, I. A. W. (September 2000). Ieee recommended practice for architectural description of software-intensive systems. Technical report, Institute of Electrical and Electronics Engineers, NY, USA.

Hilliard, R. (1999). Views and viewpoints in software systems architecture. In *First Working IFIP Conference on Software Architecture,WICSA*, pages 13–24.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., and Irwin, J. (1997). Aspect-oriented programming. In *ECOOP'97Object-Oriented Programming*, pages 220–242.

Kramer, J. Abstraction in computer science $0 software engineering: A pedagogical perspective.

Kruchten, P. (1995). The 4+ 1 view model of architecture. *Software, IEEE*, 12(6):42–50.

Majumdar, D. and Swapan, B. (2010). Aspect Oriented Requirement Engineering: A Theme Based Vector Orientation Model. *Journal of Computer Science, InfoComp*, ?(?).

May, N. (2005). A survey of software architecture viewpoint models. In *Proceedings of the Sixth Australasian Workshop on Software and System Architectures*, pages 13–24. Citeseer.

Medvidovic, N., Taylor, R. N., and Whitehead Jr, E. J. (1996). Formal modeling of software architectures at multiple levels of abstraction. *ejw*, 714:8242776.

Mili, H. and al (1999). View programming : Towards a framework for decentralized development and execution of oo programs. In *Proc. of TOOLS USA' 99*, pages 211–221. Prentice Hall.

Mili, H., Sahraoui, H., Lounis, H., Mcheick, H., and Elkharraz, A. (2006). Concerned about separation. *Fundamental Approaches to Software Engineering*, pages 247–261.

Monperrus, M., Beugnard, A., and Champeau, J. A definition of "abstraction level" for metamodels.

Mullery, G. P. (1979). CORE-a method for controlled requirement specification. In *Proceedings of the 4th international conference on Software engineering*, page 126135.

Nassar, M. (2003). VUML: a viewpoint oriented UML extension. In *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, page 373376.

Nassar, M., Anwar, A., Ebersold, S., Elasri, B., Coulette, B., and Kriouile, A. (2009). Code generation in VUML profile: A model driven approach. In *Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on*, page 412419.

Nuseibeh, B., Kramer, J., and Finkelstein, A. (1994). A framework for expressing the relationships between multiple views in requirements specification. *Software Engineering, IEEE Transactions on*, 20(10):760773.

Ossher, H. and al. (1995). Subject-oriented composition rules. In *OOPSLAS'95*, pages 235–250.

Perry, D. and Wolf, A. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52.

Regnell, B., Andersson, M., and Bergstrand, J. (1996). A hierarchical use case model with graphical representation. In *Engineering of Computer-Based Systems, 1996. Proceedings., IEEE Symposium and Workshop on*, page 270?277.

Robinson, W. N. (1990). Negotiation behavior during requirement specification. In *Software Engineering, 1990. Proceedings., 12th International Conference on*, page 268276.

Rozanski, N. and Woods, E. (2011). *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley.

Sommerville, I. (2007). *Software Engineering*. International Computer Science Series. Addison-Wesley.

Sommerville, I. and Sawyer, P. (1997). Viewpoints: principles, problems and a practical approach to requirements engineering. *Annals of Software Engineering*, 3(1):101130.

Sowa, J. and Zachman, J. (1992). Extending and formalizing the framework for information systems architecture. *IBM systems journal*, 31(3):590–616.